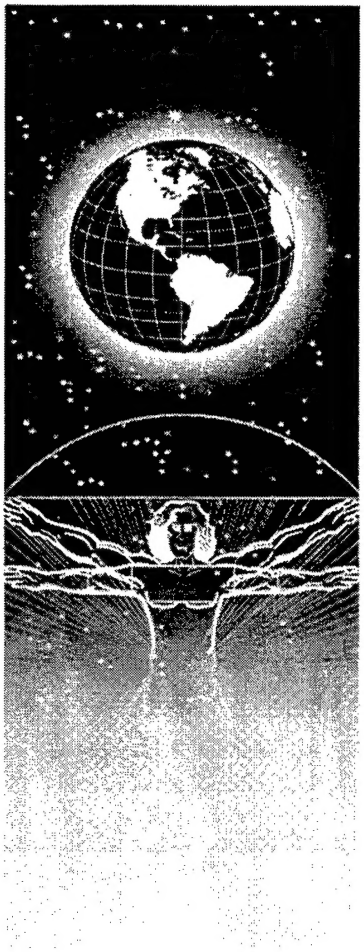
AFRL-HE-WP-TR-2000-0110



UNITED STATES AIR FORCE RESEARCH LABORATORY

Integrating Object-Oriented Simulation and Interactive Optimization Methods for Logistics Systems Analysis

S. Narayanan
Chetan Patel
Nicole L. Schneider
Hitesh Nandha

Department of Biomedical & Human Factors Engineering
207 Russ Engineering Center
Wright State University
Dayton, OH 45435

February 1997

Final Report for the Period March 1996 to February 1997

Human Effectiveness Directorate
Deployment and Sustainment Division
Logistics Readiness Branch
2698 G Street
Wright-Patterson AFB OH 45433-7604

NOTICES

When US Government drawings, specifications or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications or other data, is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Air Force Research Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161

Federal Government agencies registered with the Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
8725 John J. Kingman Rd., Ste 0944
Ft. Belvoir, VA 22060-6218

DISCLAIMER

This Technical Report is published as received and has not been edited by the Air Force Research Laboratory, Human Effectiveness Directorate.

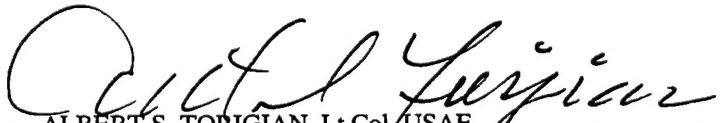
TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TR-2000 -0110

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



ALBERT S. TORIGIAN, Lt Col, USAF
Deputy Chief
Deployment and Sustainment Division
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1997	Final - March 1996 - February 1997

4. TITLE AND SUBTITLE
Integrating Object-Oriented Simulation and Interactive Optimization Methods for Logistics Systems Analysis

5. FUNDING NUMBERS
C - F41624-95-C-6014
PE - 62202F
PR - 1710
TA - D2
WU - 09

6. AUTHOR(S)
S. Narayanan, Chetan Patel, Nicole L. Schneider, Hitesh Nandha

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Biomedical & Human Factors Engineering
207 Russ Engineering Center
Wright State University
Dayton, OH 45435

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory, Human Effectiveness Directorate
Deployment and Sustainment Division
Air Force Materiel Command
Logistics Readiness Branch
Wright-Patterson AFB, OH 45433-7604

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-HE-WP-TR-2000-0110

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
User interfaces in interactive simulations convey the dynamic behavior of the modeled system and allow the analyst to interact with the executing simulation. Traditionally, the software to display the system behavior and to facilitate user interaction has been embedded in the simulation model. Such tight integrations make it difficult to maintain large simulations and impose unnecessary limitations on the development of interfaces to a simulation model. This report presents a portable, object-based architecture, called JADIS (Java-based Architecture for Developing Interactive Simulations), for developing interactive simulations. The architecture is implemented in Java and applied the Model View-Controller paradigm to the development of interactive simulations. In JADIS, the simulation model and its user interfaces are distinct processes that execute concurrently. JADIS integrates concepts from object-oriented programming, concurrent, distributed processing, and human factors interface design in developing interactive simulations. This report describes the JADIS architecture and presents its application to an aircraft repair time analysis problem in the domain of airbase logistics.

14. SUBJECT TERMS
Interactive Simulation Methodology, Object-Oriented Programming, Airbase Logistics Modeling, Java Programming Language, Model-View-Controller Framework

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

THIS PAGE LEFT INTENTIONALLY BLANK

PREFACE

This report summarizes work conducted under work unit 1710-D2-09, Optimization Method for Integration with the Integrated Model Development Environment. The work was performed under contract by Wright State University, Dayton, Ohio. Dr. S. Narayanan, was the principal investigator. Captain Todd Carrico and 1st Lt. John DiPasquale, Air Force Armstrong Laboratory, served as the Laboratory program managers. .

CONTENTS

FIGURES

THIS PAGE LEFT INTENTIONALLY BLANK

SUMMARY

This report documents the results of an approach to integrate simulation, human interaction, and evolutionary learning for solving planning problems in airbase logistics systems. Specifically, it describes the integrative approach, the design and implementation of the components of the architecture to support the integrative approach, the application of the architecture to an aircraft repair time analysis problem, and related research efforts in object-based simulations, visual interactive simulations, genetic algorithms, and interactive optimization.

The approach developed in this project is intended to facilitate computer-based aiding, where analysts can avoid the tedious work associated with the generation of alternative design configurations and focus their efforts on finding satisfactory solutions to complex problems. The two major components in the architecture are a solution explorer and an interactive analyzer. The solution explorer takes a set of design configurations as input and generates a better solution set based on genetic algorithms and an object-based discrete-event simulator. The interactive analyzer takes the better solution set as input and presents a synthetic environment to an analyst for what-if analysis and system visualization. All the components of the architecture are implemented in the Java programming language and are therefore portable across computing platforms.

The architecture is object-based and contains software abstractions for systems modeling, genetic algorithm iteration, interactive simulation development, and Pareto optimal analysis in multi-criteria decision making. The architecture is demonstrated in the context of its application to an aircraft repair time analysis problem. Results of the study are very promising. The solution explorer eliminated several inferior solutions and generated better design sets. The interactive analyzer highlighted the effect of selecting different weights to the performance measures on design alternatives and assisted in selection of the best alternative for a small-scale problem.

Further study is needed to augment these results. Application of the architecture to real world data would further facilitate evaluation of the approach and highlight strengths and drawbacks of the architecture in effectively integrating human decision making with systems modeling and evolutionary learning techniques.

INTRODUCTION

Airbase logistics is a large and complex domain involving logistics processes that support aircraft sortie generation at operational airbases. For fighter aircraft, a sortie is the takeoff, mission execution, and landing of a single aircraft. Several aircraft may be required to fly together in a mission. In part, airbase logistics involves aircraft maintenance, parts supply, and munitions loading (Popken, 1992). Models of logistics processes are useful in analysis for aircraft acquisition planning, maintenance personnel allocation, and theater-level supply redistribution.

In an airbase, there are aircraft of different kinds with varying configurations and capabilities. An aircraft is comprised of several subsystems. A typical fighter aircraft, for example, has over 300 subsystems. Sortie take-off times can be scheduled in a variety of ways from uniform random generation, flying when refueled and ready, to a specific flight schedule. Each sortie specifies the number of aircraft required, the type of each aircraft, and the details of the mission. While the aircraft is in operation one or more of its subsystems may fail. When a subsystem of an aircraft fails, it is sent to maintenance for repairs.

In a typical Air Force squadron or wing, there is a pool of personnel assigned to maintenance. Within this pool will be a variety of specialties which corresponds to areas in which the technicians are certified to perform repairs. Each repair evaluation may require aerospace ground equipment (AGE), test equipment, or special tools. Most repairs also require replacement parts.

For each configuration of aircraft, personnel, equipment, and spares, there is a wide range of sortie generation criteria that can be satisfied. The challenge is to determine if any given set of sortie requirements can be sustained for a period of time without generating excessive resource queue wait times. Each time an aircraft cannot fly due to repair problems, it is considered a maintenance abort. Models are useful to determine an efficient mix of logistics resources including spare parts, personnel, support equipment, and facilities to achieve a desired sortie rate (Boyle, 1990). Various related performance measures include maintenance cost, sorties completed, sorties aborted, facility utilization, and personnel utilization (Carrico & Clark, 1995; Carrico et al., 1995).

Two types of models have been applied extensively in logistic systems analysis: prescriptive models and descriptive models. Prescriptive models yield decisions about systems while descriptive

models evaluate the performance of a fully specified system (Dietrich, 1991). Descriptive models can also be used to analyze the impact of decisions through what-if analyses and thus, are useful as decision-making aids. Simulation is one of the most flexible descriptive modeling methods. The quality of the final solution in applying what-if analysis, however, is limited to the nature of alternatives that are examined.

Due to the complexity of the logistics domain and the qualitative nature of secondary objectives, problems such as aircraft repair time analysis are not amenable to purely prescriptive modeling methods such as math programming. What is needed is an approach that integrates simulation, evolutionary learning (Lu, et al., 1991), and human interaction (Ammons, et al., 1988) for solving design and planning problems in airbase logistics.

The overall goal of this project was to develop an approach integrating simulation, evolutionary learning, and human interaction to support generation of satisfactory solutions to complex problems in airbase logistics planning. Our approach is intended to facilitate computer-based aiding, where analysts can avoid the tedious work associated with generation of alternative designs and focus their efforts on finding good, "close to optimal" solutions to complex problems. Specific tasks include design, implementation, and evaluation of an integrated architecture to couple simulation, learning, and human interaction for an aircraft repair time analysis in the airbase logistics domain.

Section 2 describes the design of an integrated architecture. Section 3 discusses the implementation of the architecture in terms of the various components in it. Section 4 outlines the application of the architecture to an aircraft repair time analysis problem. Section 5 highlights related research efforts in interactive simulations, genetic algorithms, and human decision aiding. Section 6 presents conclusions of the study.

ARCHITECTURE DESIGN

The architecture has two major components: (1) a solution explorer, and (2) an interactive analyzer. The solution explorer is involved in the *generation of good feasible design alternatives*, while the interactive analyzer assists in the *selection of the "best" feasible alternative* by providing a synthetic environment for what-if analysis.

Figure 1 illustrates the various components of the architecture. The solution explorer has five major

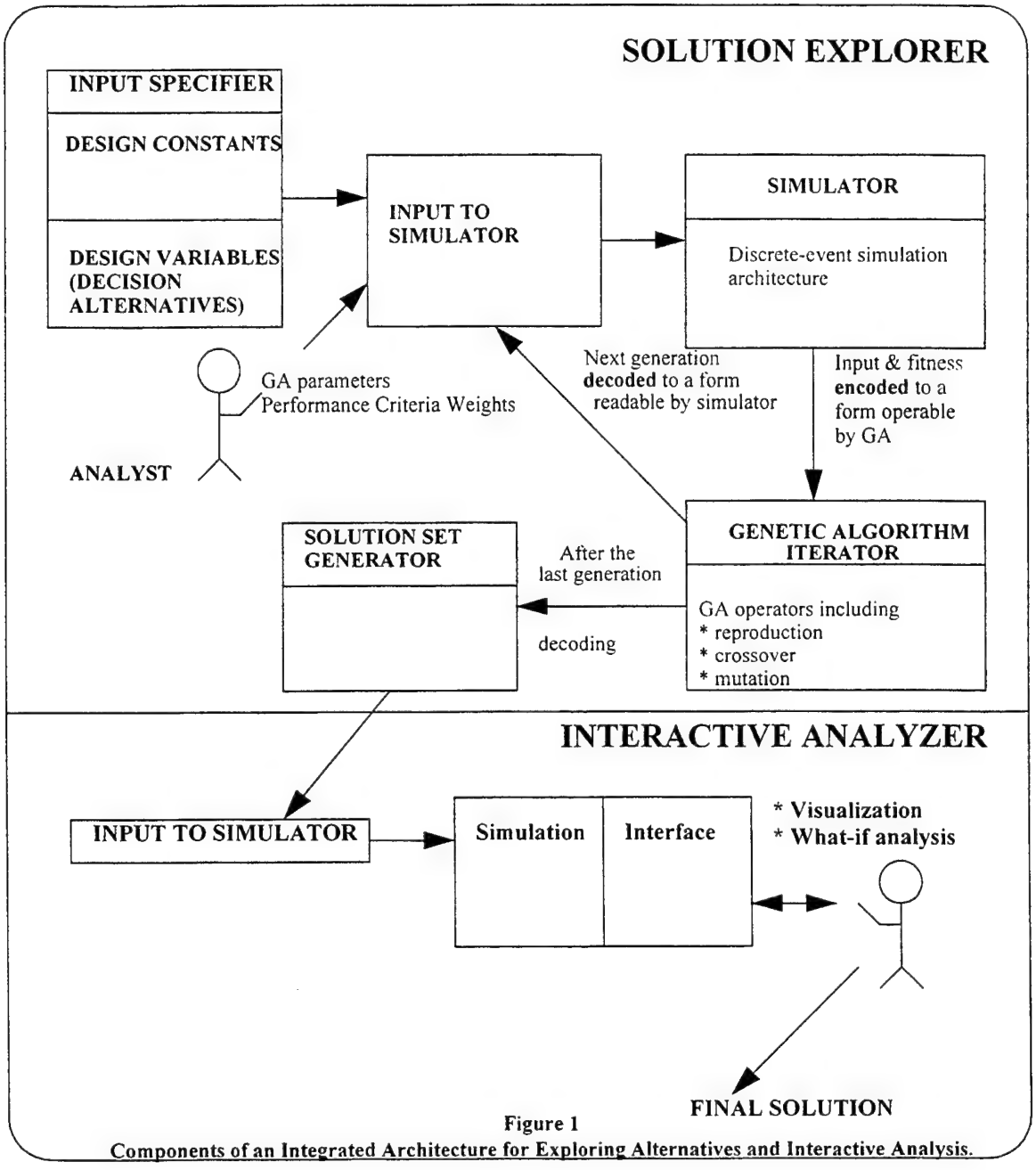


Figure 1
Components of an Integrated Architecture for Exploring Alternatives and Interactive Analysis.

modules: (1) input specifier, (2) simulator, (3) encoder/decoder, (4) genetic algorithm iterator, and (5) final solution set generator. The solution explorer is initiated by a human analyst. The analyst specifies a set of initial solutions that are to be used by the explorer to generate better solutions. Part of the input specification is constant whereas the remaining portion constitutes decision variables that can be changed during the solution exploration phase. The analyst also specifies the relative importance of various performance measures in the system as well as parameters for the genetic al-

gorithm iterator. The simulator encapsulates a descriptive model of the system for which the alternatives need to be explored. The encoder transforms the simulation input to a form operable by the genetic algorithm iterator and the decoder performs the reverse process. The genetic algorithm iterator performs evolutionary learning through random, yet directed, search of the solution space through the application of operators such as reproduction, crossover, and mutation (Goldberg, 1989). After the specified number of iterations through the genetic algorithm iterator, the decoded output from the genetic algorithm iterator comprises the final solution set that can be examined by the interactive analyzer.

The interactive analyzer consists of an interactive simulator based on the Model-View-Controller paradigm (Goldberg, 1990; Krasner & Pope, 1988). The solution set generated by the solution explorer can be thoroughly examined by a human analyst through what-if analysis and visualization of the solutions. Analysts can generate a Pareto optimal solution for multi-criteria decision making problems and also make real-time decisions such as modifying scheduling disciplines or altering maintenance resources in the context of a running simulation. Through analysis, the human selects the best feasible alternative for the specific design and planning problem.

Our approach attempts to couple computational tools with human intelligence in a single integrated system that maximizes joint performance (Woods, 1986). Our approach is based on several assumptions. First, it is assumed that several feasible solutions exist for a given design or planning problem. Second, our approach assumes that humans are better at examining a few different alternatives rather than in the generation of several alternatives to the design problem (Brill, et al., 1990). Third, it is assumed that the decision variables in the system can be represented in a form for which genetic algorithms can be applied. Fourth, our approach assumes that closed-form analytical solutions do not exist for the problem for which the approach is being applied and that analysts rely primarily on heuristic decision making and both quantitative and qualitative criteria in decision making.

ARCHITECTURE IMPLEMENTATION

All components of the architecture are implemented in the Java programming language. Java is an object-oriented programming (OOP) language whose syntax is similar to C++ and supports encapsulation, inheritance, and polymorphism. A glossary of terms commonly used in OOP is included in the Appendix. Java has become a popular programming language on the Internet due to applets.

Applets are Java programs that can be embedded on home pages on the world wide web. Java can also be used as a standard programming language, constructing stand-alone applications independent of web pages. The Java language contains various packages (similar to software libraries) for general data structures, applets, file input/output, and also for graphical user interfaces. Java is multithreaded and hence particularly suitable for distributed computing (Niemeyer & Peck, 1996). Java can be used for both creating simulations as well as for creating interfaces to simulations unlike languages such as C++, where we need to use X windows, Motif, or other external software libraries for interfaces. A major advantage of using Java is its portability. Typically, source code written in Java is compiled into byte code that can be read by an interpreter available on multiple platforms including personal computers, Macintoshes, and UNIX workstations. The software can be developed on any platform that contains the Java Development Kit (JDK). JDK is available on most operating systems including `Solaris, Windows 95, Windows NT, and MacOS`. The Java compiler in JDK generates byte code from the Java source code. The byte code can then be moved to other platforms and can be executed without altering the code. Figure 2 outlines the Java virtual machine environment.

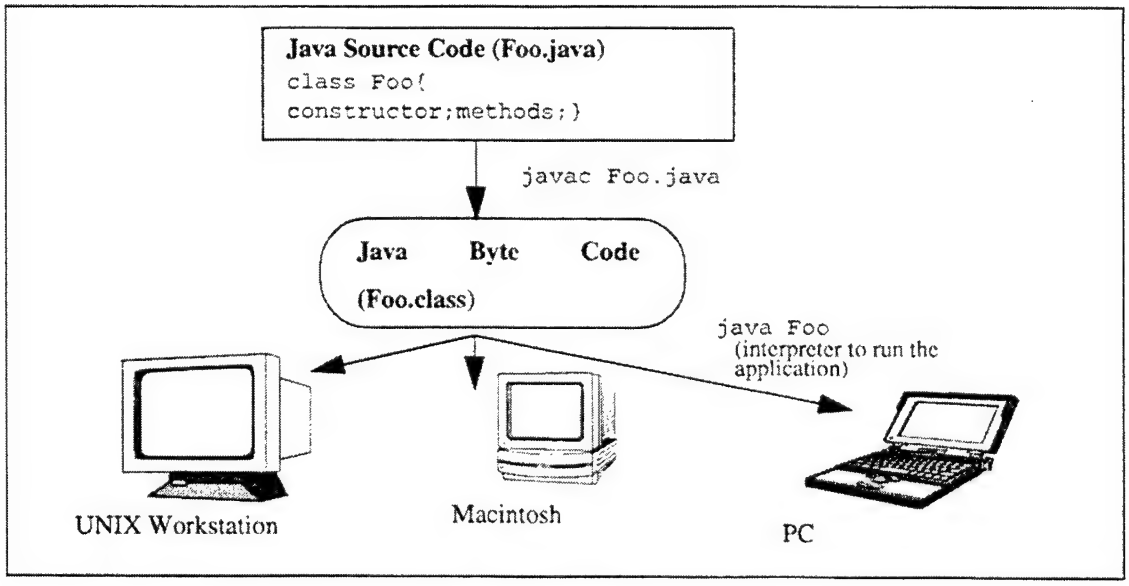


Figure 2
Illustration of the Java Virtual Machine Environment.

The implementation of the various modules of the two components of the architecture: (1) solution explorer and (2) interactive analyzer are described below.

Input Specifier

The input specifier module is used by a human analyst to specify a set of input configurations that the solution explorer takes in as initial solutions. The input is in the form of ASCII text files, where the design parameters are broken into constants and decision variables. The variable part of the design input gets changed during the iterative process. An analyst also specifies the weights of the various performance criteria in the system and also inputs the parameters of the genetic algorithm iterator (e.g., number of iterations). Figure 3 illustrates a partial input file consisting of a few design constants and several decision variables in the context of an aircraft repair time analysis problem. In the example shown, three major parameters including number of aircraft, personnel data, and equipment data are the decision variables.

File containing design constants

```
# SIMULATION INFO

simEndTime 100
simStartTime  0

# SCHEDULER INFO
schedulerType RANDOM
generationRate seed 102567
generationRate upperBound 5.0
generationRate lowerBound 1.0

# RANDOM SCHEDULE INFO
takeOffTime seed 125469
takeOffTime lowerBound 2
takeOffTime upperBound 4
flightDuration seed 427689
flightDuration lowerBound 1
flightDuration upperBound 4

# SUBSYSTEM INFO
52 subCount 3
Door
Engine
Brake
```

File containing decision variables

```
numParams 3

# AIRCRAFT DATA
numAircraftType 2
52 numAirCraft 10
53 numAirCraft 15

# PERSONNEL DATA
numPersonnelType 2
FFSC numPersonnel 4
FSC numPersonnel 5

# EQUIPMENT DATA
numEquipmentType 2
MC-1A numEquipment 2
N2-CART numEquipment 2
MC-1A numEquipment 1
N2-CART numEquipment 2
AM32C-10 numEquipment 3
```

Figure 3
Partial Input File Containing Design Constants and Decision Variables for Aircraft Repair Time Analysis.

Simulator

The simulator module in the solution explorer consists of the application-independent components of the simulation infrastructure including a random number generator, various standard statistical distributions (such as `Uniform`, `Exponential`, `Normal`, and `LogNormal`), an event calendar which keeps track of future events, a simulation clock, and general queueing support.

The simulator also contains classes to represent entities and their interrelationships in the application domain. The development of the Java classes to model the airbase logistics processes are based on three design principles. First, the simulation objects should have a direct correspondence to the resources, material, and control processes in the airbase logistics domain. Object-oriented programming (OOP) enables a developer to build software abstractions in simulations that have a direct correspondence with real world objects (Narayanan et al., 1996). In our architecture, we exploited the natural mapping and modularity features of OOP and developed a large number of Java classes such as `Aircraft`, `Hangar`, `Spares`, and `Equipment` that have a direct mapping to real world entities and logistic processes. The second design principle is to make objects that represent decision making distinct from physical objects and information storage objects. In applying JADIS (Java-based Architecture for Developing Interactive Simulation) to airbase logistics, physical objects such as `Aircraft` are distinguished from decision making objects (e.g., `ResourceManager`) and information storage objects (e.g., `ResourceStatistics`). Simulations facilitate the study of complex systems, by decomposing the decisions, the data used by the decision logic, and the results of the decisions on the physical system in a structured manner. The approach of making the physical-control-information decomposition has been successfully applied by several researchers in object-oriented simulations of large manufacturing systems (Adiga & Glassey, 1991; Mize et al., 1992; and Narayanan et al., 1992 & 1996). Hence, applying a similar principle in modeling large, complex, and dynamic airbase logistics systems is appealing. The final design principle is related to rapidly assembling simulations of a system for different configurations. A major goal of the solution explorer is to represent alternative system configurations. Therefore, the hierarchy of classes and the implementation of the classes are designed to facilitate rapid assembly. The classes are designed to be modular and extensible. Detailed descriptions of specific classes implemented for aircraft repair time analysis is described in Section 4.

Since the goal of the solution explorer is to represent alternate designs and there is little human interaction after the initial parameters are specified, the simulator module does not have a graphical

user interface to depict the system dynamics. The performance measures are gathered during the execution of the simulation program, user-specified weights are applied to generate the fitness value of the specific input configuration, and the relevant values are communicated to the genetic algorithm iterator.

Encoder/Decoder

The encoder translates the variable decision alternatives part of the design input into a form that can be used by the Genetic Algorithm iterator. Each input set is represented by a string called genome. Each genome is in turn made up of subunits called traits. Each trait stores an encoded value of an input set parameter. The genome also stores a fitness value associated with the input set. The fitness value is generated by the simulator. The process associated with the encoding/decoding mechanism is dependent on the specific problem being analyzed. In our design, the genome and traits can be of variable lengths. The parameter values are encoded in the octal format. The encoding process converts integers and floats into octal representations. The trait also contains a *key* that will indicate the number of digits used to encode each part of the trait. Figure 4 illustrates the encoding process for a partial input set. The decoding process is a reverse of the encoding mechanism. The example shows the encoding in the octal format for three parameters and nine values. The final input string forms an input sample in the population that evolves in the GA iterator.

```
#  INPUT SET                                          #  ENCODED VALUE

# PARAMETER NAME                VALUE
#
# AirCraft Data
airBase-52 numAirCraft            10                      12
airBase-53 numAirCraft            15                      17

# Personnel Data
airBase-FFSC numPersonnel          2          Encoding     02
airBase-FSC  numPersonnel          3          ------->     03

# Equipment Data
airBase-AM32C-10 numEquipment 2               Decoding     02
airBase-AM32A-10 numEquipment 2              <-------      02
airBase-MC-2A    numEquipment 2                            02
airBase-MC-1A    numEquipment 2                            02
airBase-N2-CART  numEquipment 2                            02
```

FINAL INPUT STRING: 121702030202020202
(The final input string shown above forms a sample in the population.)
KEY: 2 (indicates number of digits per parameter)

Figure 4
Illustration of the Encoding Process for a Partial Input Set in Airbase Simulation.

Genetic Algorithm Iterator

The genetic algorithm (GA) iterator applies random, but directed search algorithms based on the mechanics of natural selection and genetics. GAs are particularly useful for complex systems where traditional analytical techniques such as calculus-based and enumerative techniques fair poorly. GAs have been successfully applied in many applications in function optimization, image processing, engineering, and business (Goldberg, 1989). Four major differences exist between the conventional optimization techniques and GA based optimization. First, GAs work on the direct manipulation of a coding of the decision variables unlike traditional methods which usually deal with functions and control variables directly. Second, GAs work from a population rather than a single point solution used by many traditional methods. Thus, GA-based search methods are suitable for situations where local optima exist. Third, the search process in GA is via sampling from the population and primarily relies on the payoff from solutions set. Traditional methods rely heavily on application-specific information. Thus, GAs are more easily applicable to situations where the mathematical formulation of the problem is hard. Finally, the search in GAs is based on stochastic operators, not deterministic rules as used in simple random walks, making GA based search process highly exploitative.

In our architecture, the GA iterator is developed along the same lines as GAlib, a C++ library of GA components developed by Matthew Wall at MIT. The GA iterator in our architecture is implemented in Java and is tailored to airbase logistics problems. Major classes implemented include `Genome` (contains an input set string), `Trait` (stores trait values for each genome), `Population` (encapsulates population information), `GAStatistics` (stores statistics generated during the GA process), `GAScaling` (represents the scheme to perform fitness scaling), `GAMutationScheme` (defines the scheme for mutation), `GASelectionScheme` (has the scheme for reproduction of genomes), and `GACrossoverScheme` (has the details of the crossover operation). The fitness values of input sets are obtained from the simulator and is a function of the various performance measures. The population size is assumed to be fixed. We apply three GA operators: reproduction, crossover, and mutation. The parameters of the GA are user specified. The crossover operation takes place in two parts. First, the specific trait in the genome where the crossover occurs is determined. Then, the specific part of the trait to crossover is determined. Reproduction and mutation operation occurs like in the simple GA mechanism described in Goldberg (1989). Through several iterations of the GA operators, the solution space is explored.

Final Solution Set Generator

After the user-specified number of iterations are completed, the population set is decoded by the final solution set generator to form the set of solutions that are to be examined by the interactive analyzer. The decision variables are assembled with the design constants to form solutions to the design problem.

Interactive Analyzer

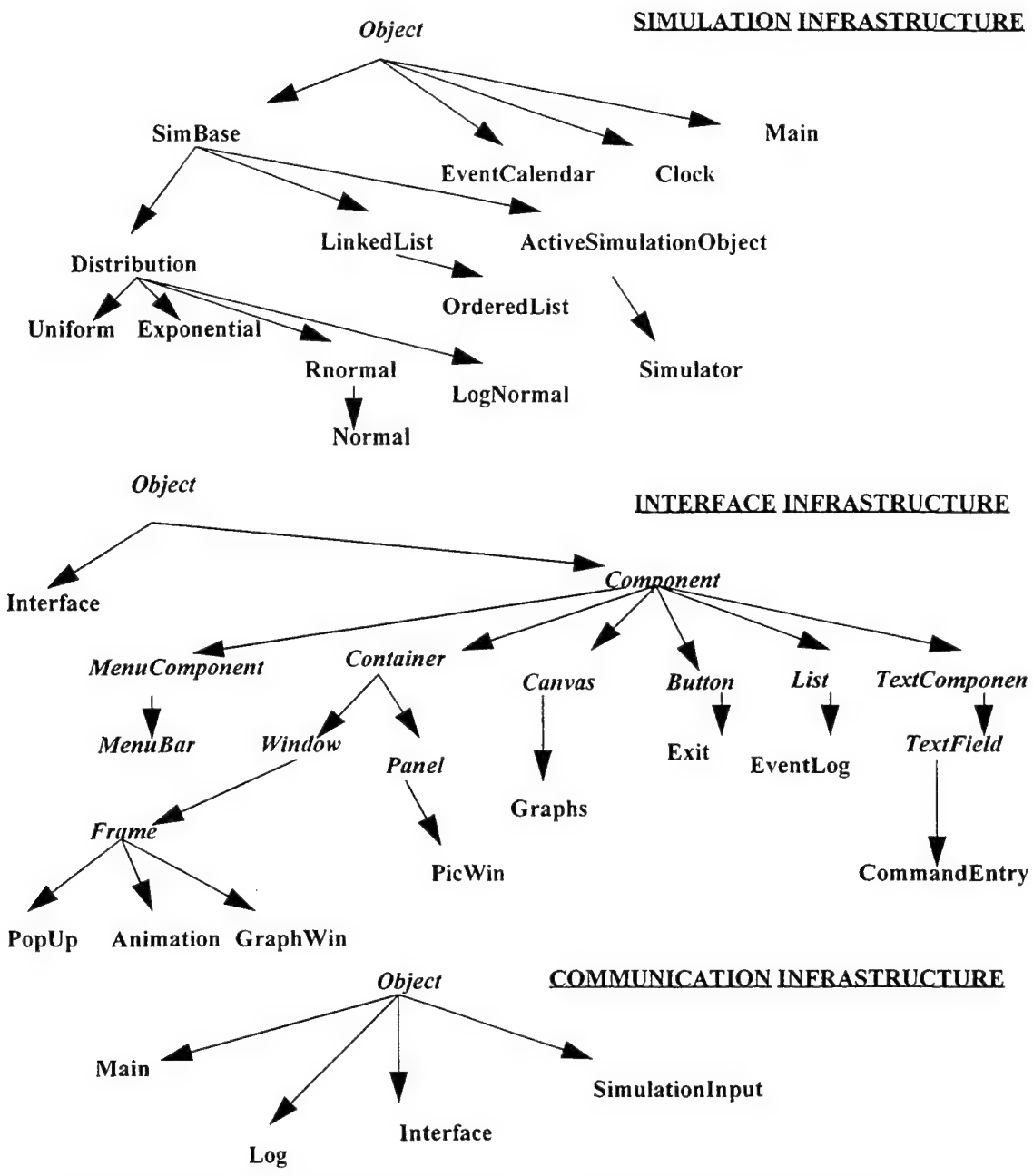
The interactive analyzer enables the human analyst to closely examine a few good, feasible solutions generated by the solution explorer. The interactive analyzer contains capabilities for interactive simulation as well as visualization of results.

Interactive Simulator

Our integrated architecture contains a portable, object-based simulator called JADIS for interactive simulations. User interfaces in interactive simulations convey the dynamic behavior of the modeled system and also allow the analyst to interact with the executing simulation. JADIS integrates concepts such as Model-View-Controller, concurrent, distributed processing, and human factors interface design in developing interactive simulations. An introduction of interactive simulations and details of JADIS can be found in Narayanan et al. (1997). A brief overview of JADIS is provided below.

In JADIS, the simulation and user interface are two distinct processes. JADIS contains infrastructure for simulation, interface, and inter-process communication. The main simulation loop in JADIS can operate in real-time or next event time transition mode. User input can be obtained through interaction at the interface even while the simulation continues to execute. Figure 5 illustrates the various application-independent software abstractions in JADIS.

All classes in JADIS are subclasses of the base Java class `Object`. In addition, there is an abstract class, `SimBase`, from which several of the simulation infrastructure classes are inherited. Class `Distribution` encapsulates a portable random number generator using the linear congruential method (Law & Kelton, 1991). Classes `Exponential` and `Uniform` use the inverse transformation method in generating exponential and uniform distributions respectively. Class `Rnormal` gen-



Note: The italicized classes are part of the Java language (Not all classes are shown).

Figure 5

Application-Independent Software Abstractions in JADIS.

erates a standard normal distribution using the composition method described in Brantley, Fox, and Schrage (1987, p.318). Class `Normal` generates a normal distribution with a given mean and variance by transforming the standard normal. Class `LogNormal` generates a log normal statistical distribution using the inverse transformation method outlined in Law and Kelton (1991, pp. 259-260).

Class `Clock` encapsulates the simulation clock. `Clock` has an `updateClock` method which up-

dates the simulation clock to the next event time if the simulation is not operating in real time or updates the clock in microseconds corresponding to the computer system clock. Classes `LinkedList` and `OrderedList` are queueing utilities available in JADIS. Class `EventCalendar` keeps track of the details of the events including information about the event ordered by the time when it needs to be executed. Events which are placed on the `EventCalendar` must be encoded as methods of class `ActiveSimulationObject` or its subclasses. Class `Main` encapsulates the main simulation loop in JADIS. The main loop generates a thread to obtain input from the interface process and creates another for the simulation process to continue to execute. The main simulation loop invokes an event from the `EventCalendar` at the scheduled time and updates the simulation clock appropriately. Class `Simulator` is a subclass of `ActiveSimulationObject` and has methods to initialize the simulation parameters and exit at the end of the simulation.

The abstractions comprising the interface infrastructure include the main interface process and building blocks such as windows, menus, scrolling lists, pop-up boxes, and push buttons which are useful in developing an interface to an interactive simulation. Most of these abstractions are subclasses of the elements of the abstract windowing toolkit that is part of the Java language. As shown in Figure 5, JADIS classes in the interface infrastructure include `CommandEntry` useful for gathering user commands to the simulation through a text-based command line, `EventLog` is a scrolling list to output a log of the events occurring in the simulation, `Exit` is a push button to halt the simulation and close the interface, `PicWin` is useful to display icons of the simulated system, `PopUp` is useful to implement windows that pop up as a result of user action on the interface. Class `MenuBar` is part of the Java language useful for developing menus to the interface. Classes `GraphWin` is window used to display graphs of the simulated system's performance measures with class `Graphs` providing the drawing area canvas. Class `Animation` assembles all the necessary interface building blocks for a simulation. `Animation` has a method called `processEvent` which in turn invokes a `processEvent` method of displayed objects. Class `Interface` initiates the interface process and spawns another process to concurrently receive input from the simulation model. Users of JADIS typically create additional subclasses of the classes in the interface infrastructure to tailor it to the application domain. Interface classes created for the domain of airbase logistics are outlined in the next section.

As shown in Figure 5, four major classes play an integral role in the communication infrastructure.

They are `Main` and `Log` in the simulation side and `Interface` and `SimulationInput` on the interface side. Main spawns two threads, one for the simulation and another to receive input from the interface process. It also has a parser to identify the command received from the interface process. Messages are in the form of strings. Messages from the interface process to the simulation are of the form "commandName commandDetails." For example, when the user has clicked on the exit button on the interface, the interface process sends the message "exit" to the simulation. Similarly, when the user has increased the speed factor of the simulation clock to 5, the interface would send the message "speed 5." Class `Log` initializes a socket connection between the simulation process and the interface process and has a method `sendEvent` to send a message from the simulation to the interface. Messages from the simulation to the interface are of the form "ObjectName EventName EventDetails." For example, in the simulation of aircraft maintenance operations, a message "aircraftb52-10 landed" is sent when a specific aircraft object lands after performing a mission. Class `Interface` spawns the process to display the simulated system and spawns another process to receive input from the simulation. Class `SimulationInput` establishes a socket connection from the interface process to the simulation and has a method called `SendCommand` that sends messages from the interface to the simulation. The communication infrastructure in JADIS is flexible and modular to accommodate multiple views to the same simulation model. The application-dependent portion of the interactive simulator is described in section 4.

Visualizer

The interface in the interactive analyzer serves to display system dynamics and to facilitate user interaction. Graphical output of relevant performance measures are available at run time. Built-in capability exists for visualizing the logistic processes in aircraft maintenance. Users can also alter information on maintenance operations and dynamically alter scheduling strategies while the simulation continues to execute. Several runs of the simulation are used for alternative designs. For each design, the mean, variance, and confidence interval is calculated. Analyst can then query the system for the Pareto optimal solution. The visualizer identifies the Pareto optimal for the user specified multiple objectives and presents the solution set pictorially. An analyst can also alter weights of the performance measure and investigate the effects of criteria on the overall design. Throughout the entire process, the architecture keeps track of the design input and associates it with the system performance. The next section describes the application of the architecture for an aircraft repair time analysis problem.

ARCHITECTURE APPLICATION

The architecture was applied to model a prototypical airbase facility. Maintenance operations for a two week period were simulated with sortie generation occurring 16 hours a day for seven days a week. Figure 6 highlights Java classes in the simulator (both in the solution explorer and interactive analyzer) to represent entities and their interaction for aircraft maintenance. Objects that encapsulate events in the simulation are subclasses of `ActiveSimulationObject`. These include physical objects such as `Aircraft`, `Subsystem`, `Spare`, `Sortie`, `Airbase`, and `Resources` such as `Equipment`, `Hangar`, and `Personnel`. The physical objects have a one-to-one mapping to objects in the real world. Class `Aircraft` is a representation of aircraft in an airbase and is comprised of several `Subsystems`. Class `Spare` represents consumable spare parts used during maintenance. Class `Sortie` models sorties generated during a time horizon. Class `Airbase` is a collection of several entities including `Aircraft`, `Spares`, and `Resources` such as `Equipment`, `Hangar`, and `Personnel`. Class `Equipment` represents aircraft ground equipment needed for maintenance operations. Class `Hangar` represents a hangar in an airbase and class `Personnel` represents maintenance operators.

Class `DecisionMaker` and its subclasses encapsulate logistical decision making. Subclasses of `DecisionMaker` include `ResourceManager`, `Scheduler`, `FailureGenerator`, and `Coordinator`. The `Scheduler` generates sorties for an airbase. The `FailureGenerator` enables subsystems in aircraft to fail according to their failure behavior. The `ResourceManager` controls the handling and distribution of all resources including equipment, hangar, and personnel needed for maintenance. The `Coordinator` is an overall airbase controller responsible for controlling the movements of various physical objects in an airbase. There is at least one instance of all the decision making classes in any simulation of airbase maintenance operations.

Class `InformationStorage` and its subclasses encapsulate data associated with the simulation. Subclasses of `InformationStorage` include `ResourceStatistics`, `MaintenanceInfo`, `AircraftStatistics`, `SubsystemStatistics`, and `ScheduleInfo`. Class `MaintenanceInfo` encapsulates data associated with the maintenance operations including type of failure, repair time, and the necessary type and number of personnel, spares, and equipment. Class `ScheduleInfo` encapsulates information associated with the sortie generation. There are three subclasses of `ScheduleInfo` (not shown in Figure 6): `SetScheduleInfo`, `Random-`

ScheduleInfo, and FlyWhenReadyScheduleInfo, each encapsulating sortie information according to the appropriate mode of sortie generation. Classes ResourceStatistics, AircraftStatistics, and SubsystemStatistics all encapsulate statistical information of relevant performance measures gathered during the execution of the simulation program.

Figures 7 and 8 depict the input screens used by an analyst in specifying the relevant performance measures and their relative weights. Most of the GA parameters including probability of cross over and mutation are read from data files. Once these inputs are specified, there is little interaction between the architecture and the human analyst during solution exploration. Figure 9 presents a snapshot of the GA-based iterative search process.

After the specified number of iterations, the final solution set generator outputs a set of solutions to be examined by the interactive analyzer. Figure 10 is the main interface window for the interactive analyzer for the airbase simulation. All the visible objects are instances of the basic interface infrastructure in JADIS. The interface has a MenuBar with five menu items, four PicWin windows depicting graphical icons of the facility, an EventLog displaying a scrolling list of events occurring on the simulation side, a CommandEntry window through which users can enter commands to the simulation through a keyboard, and a Exit button to halt the simulation and exit the process.

Graphical outputs of various relevant performance measures are available at run time and are presented through instances of GraphWin and Graphs classes. Figure 11 illustrates graphs during a snapshot of the system for three performance measures including sorties completed, equipment utilization, and personnel utilization. The graph on the right hand side shows greater detail of the performance measure by depicting values of the individual parameters. An additional class called Dynamics was implemented as a subclass of Frame. Dynamics is useful to display visualization of the processes involved in airbase maintenance. Figure 12 illustrates a dynamics window which shows the number of aircraft waiting for take off, in flight, waiting for maintenance, and in maintenance during a time in the simulation. Through mouse clicks, a user can alter information on maintenance operations and also dynamically alter the mode of sortie generation even while the simulation continues to execute.

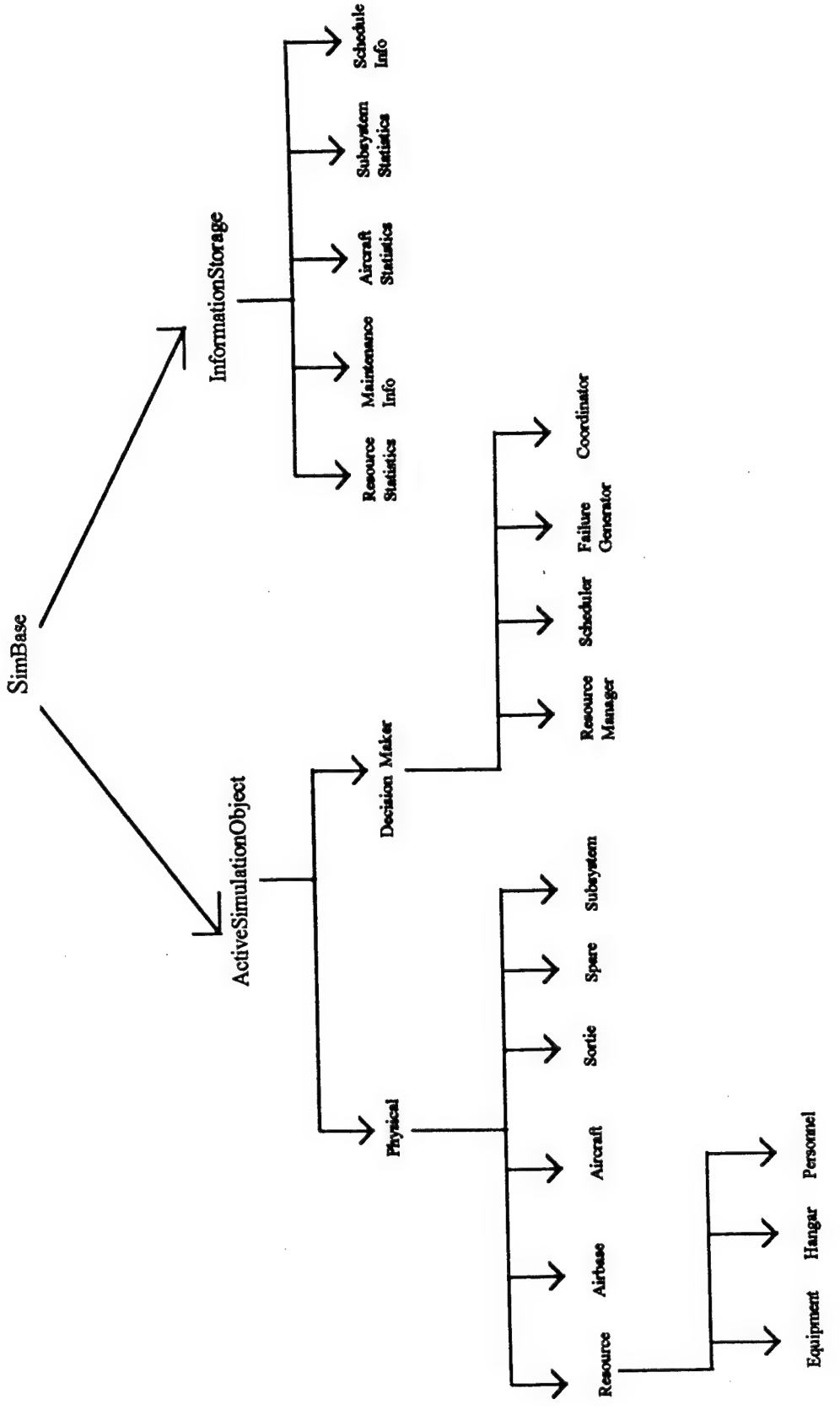


Figure 6
Hierarchy of Salient Classes in the Simulation of Airbase Logistics Domain.

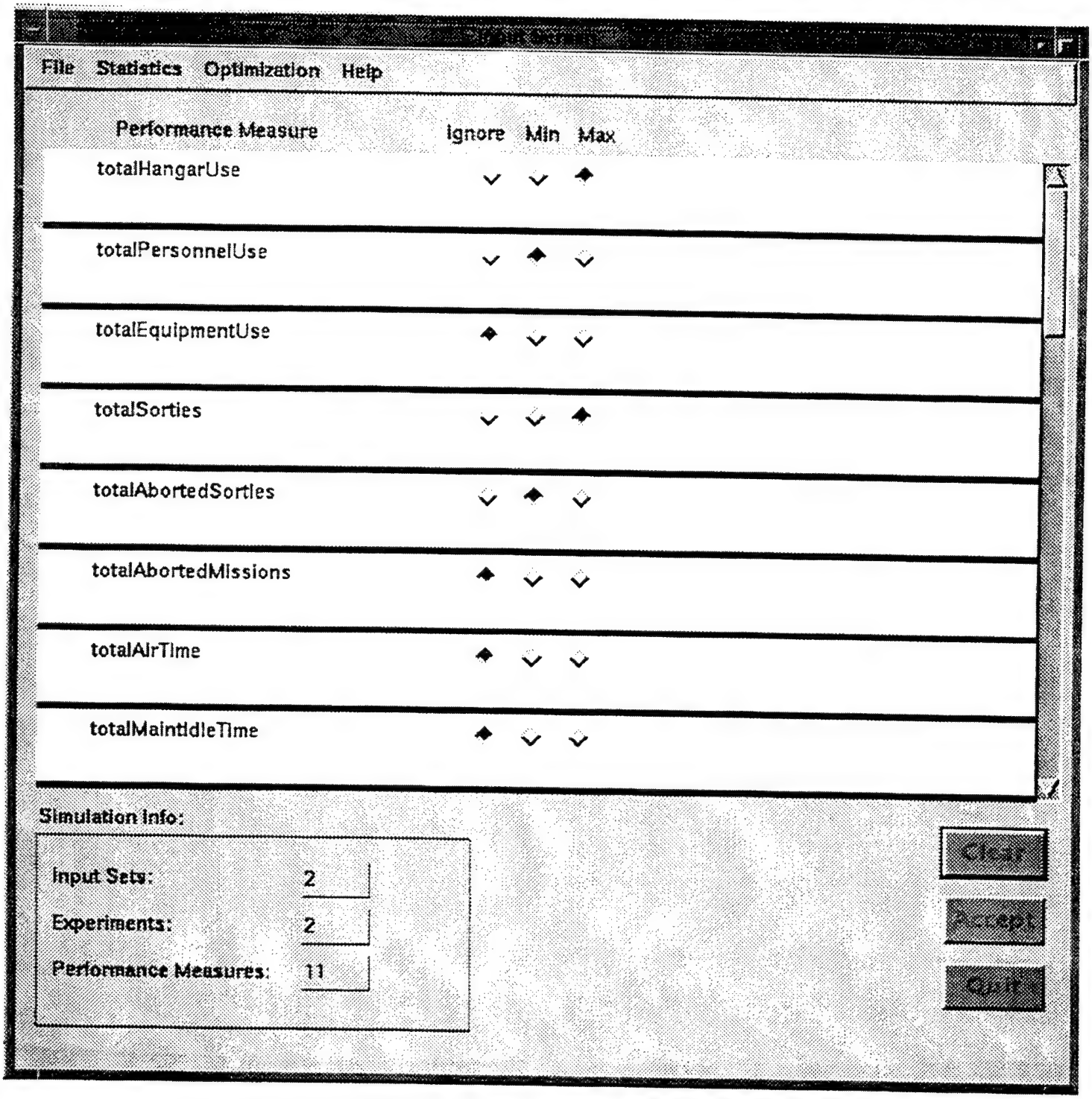


Figure 7
Interface for the Solution Explorer to Obtain Input on Performance Measures.

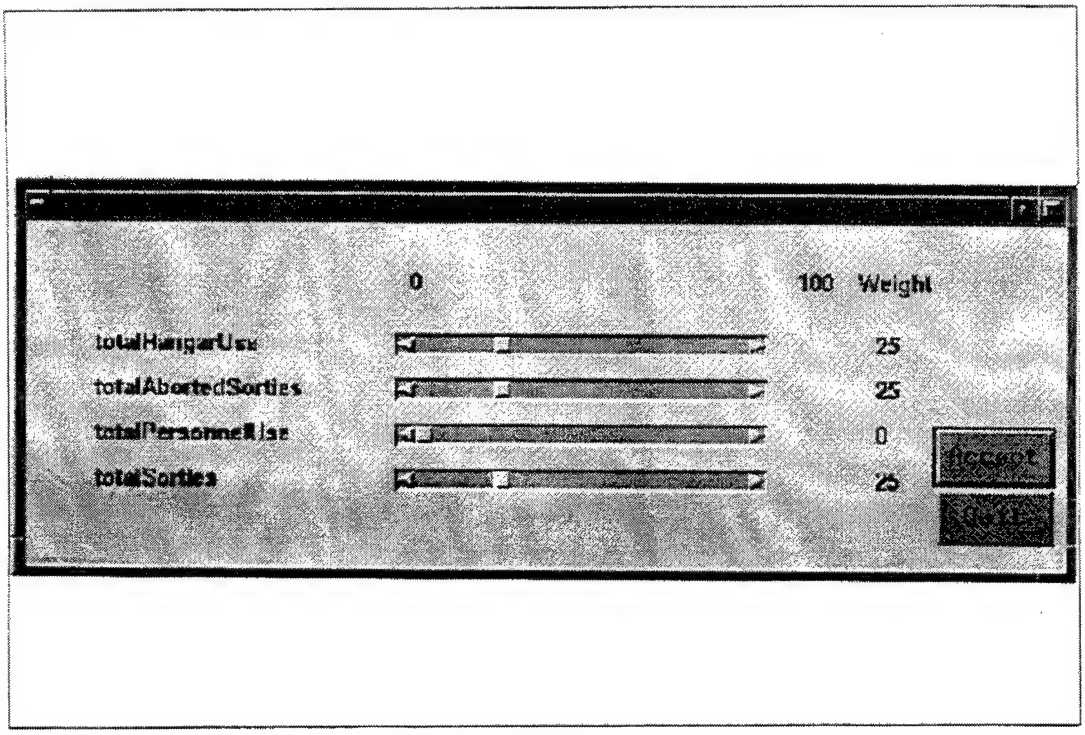


Figure 8
Input Screen for the User to Specify Weights to the Different Performance Measures.

The interactive analyzer stores relevant information for several runs of each design. The data on the performance measures collected during simulation are used by the analyzer to generate mean, variance, and confidence interval at the user-specified alpha level. These values are used to identify the Pareto optimal solution. In a Pareto optimal set, the dominated solutions are eliminated from further consideration. An analyst can specify different weights for the performance measures and can use the interactive analyzer to examine the effects on the alternatives. Figure 13 presents a pictorial representation of a Pareto optimal solution for an aircraft repair time analysis problem.

Thus, our integrated architecture is useful for simulation, solution exploration, search space pruning, and evaluation of solutions in multi-objective problems. The next section presents an overview of related research efforts.

```
Initializing AirCraft, Personnel, Equipment, Hangars....

Executing Simulation...
fitness of design0 is 0.71
fitness of design1 is 0.6
The best design no is 0
Performing GA operations...
raw score read from output file: outset0 is 0.71
raw score read from output file: outset1 is 0.6
Fitness Statistics for the current generation:
Minimum Fitness 0.6
Maximum Fitness 0.71
Average Fitness 0.65
Fitness Sum 1.31
Encoding Genome1:
StringValue: 121702030202020202
Fitness Value: 0.71
Raw Scores: 0.65
decodeFile: inpset0
encodeFile: outset0
Encoding Genome2:
StringValue: 010203030101020201
Fitness Value: 0.6
Raw Scores: 0.6
decodeFile: inpset1
encodeFile: outset1
```

Figure 9
Illustrative Example of GA-based Iteration.

RELATED RESEARCH

Several areas including object-based simulations, visual interactive simulations, genetic algorithms, and interactive optimization are relevant to the research effort described in this report. A brief overview of the relevant research areas are reviewed in this section in the context of the results of this study.

Object-Based Simulations

Recently, there has been a growing interest in object-oriented programming (OOP) applied to simulation modeling of complex systems. Modular design, software reusability, potential for natural mappings, and compatibility between the OOP paradigm and the discrete-event world view formalism are primary reasons for this interest. BLOCS/M (Adiga & Glassey, 1991), DEVS (Zeigler, 1991), OOSIM (Narayanan et al., 1992, 1996), and OSU-CIM (Mize et al., 1992) are examples of

Figure 10
Main Interface Window for the Interactive Analyzer.

Figure 11
Graphical Output of a Snapshot of Airbase Simulation.

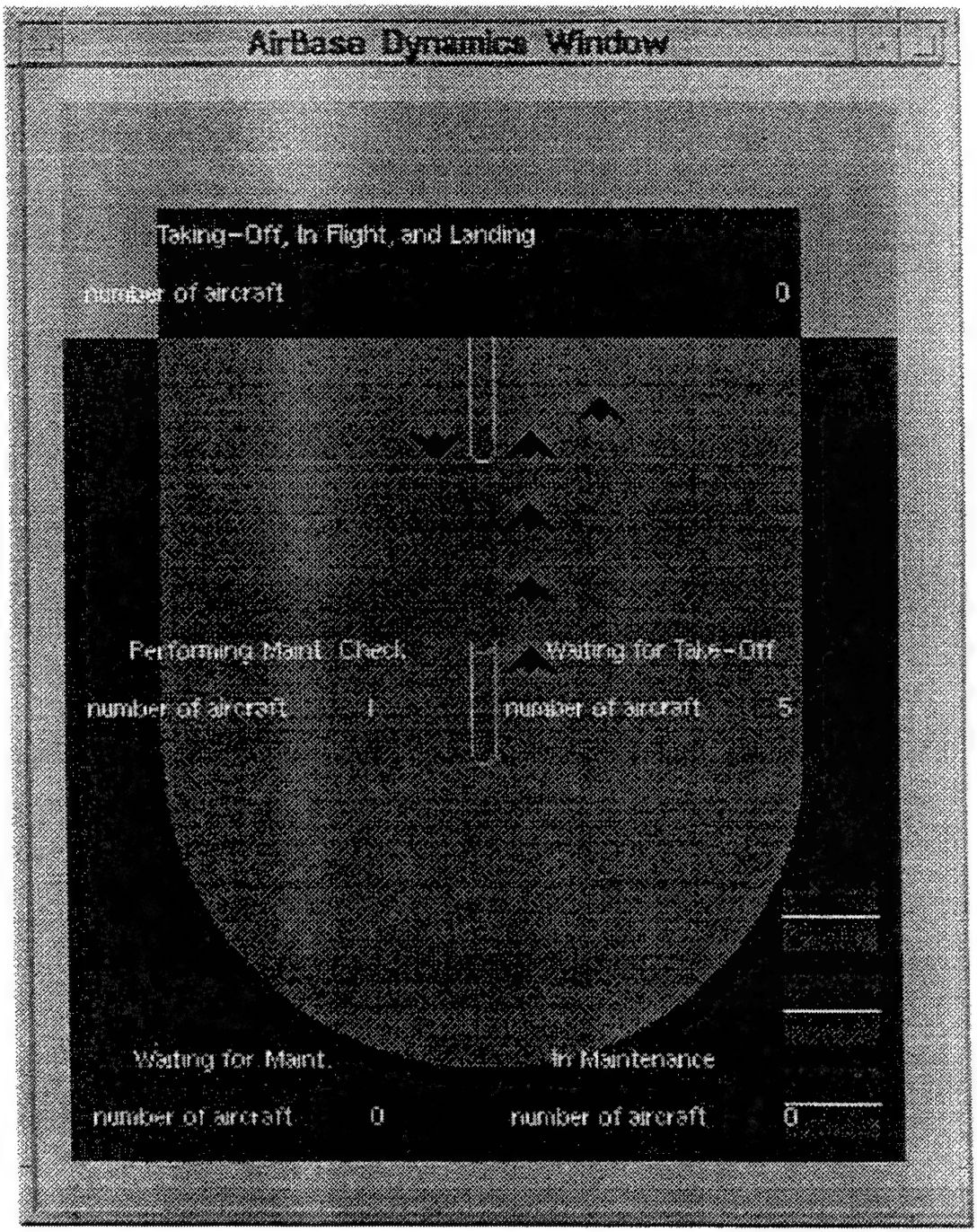


Figure 12
Visualization of the Dynamics in Airbase Simulation.

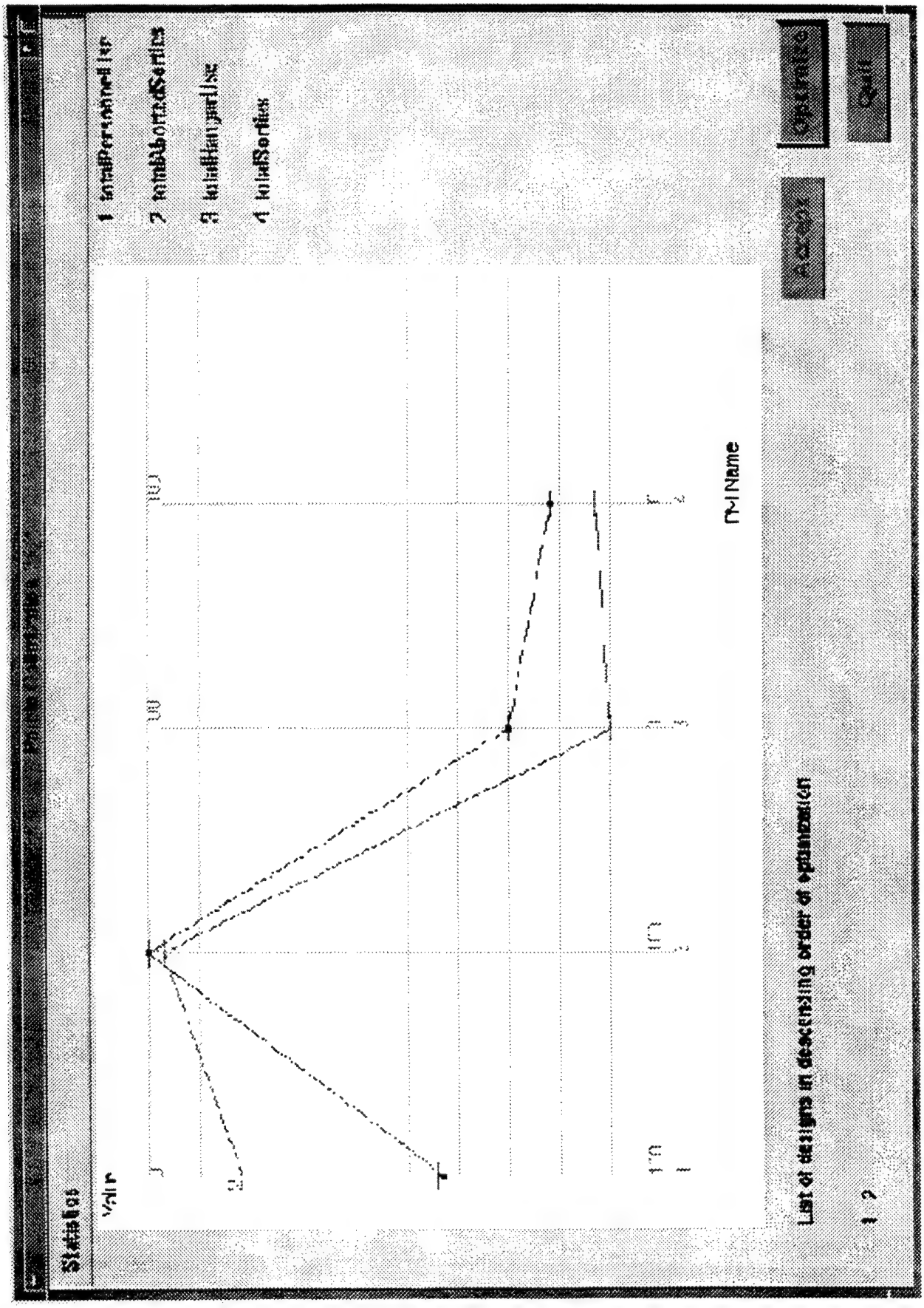


Figure 13.
Pareto Optimal Solution for an Aircraft Repair Time Analysis Problem.

several object-based simulation architectures for modeling systems in the manufacturing domain. IMDE is an object-based architecture developed for airbase logistics simulations (Carrico & Clark, 1995; Carrico et al., 1995). The object-based design of the simulator in the architecture we have developed is also motivated in part by considerations of natural mapping, modularity, and software reusability. In addition, since our architecture is implemented in Java, it is highly portable. We have also made a distinction between decision-making objects, physical objects, and information storage objects which facilitates modular representation. The simulator in the interactive analyzer component contains objects to develop graphical user interfaces in addition to objects for simulating systems. Most simulation architectures described in the literature do not have a built-in architecture for developing graphical interfaces. Perhaps the most significant difference between our architecture and the other efforts in object-based simulation research is the incorporation of the solution explorer component through genetic algorithms and the accommodation of human interaction in the interactive analyzer component in the selection between alternatives. In many of the other object-based simulation efforts, the simulations are descriptive and the alternatives to be evaluated have to be specified manually.

Visual Interactive Simulations

With the advances in computing power and graphical user interfaces, there is an increasing interest in the area of visual interactive simulations (Bell & O'Keefe, 1987; Bell, 1991; Hurrion, 1980; Lyu & Gunasekaran, 1993; McGregor & Randhawa, 1994). In visual interactive simulations (VIS), interfaces serve to not only display the state of the simulated system, but also to allow an analyst to interact with the executing simulation. As the simulation executes in real time or scaled time, the analyst can modify the parameters and alter the dynamics of the simulated system.

The VIS approach offers several potential advantages. First, it allows the user to interactively make complex decisions. For example, Hurrion and Secker (1978) found that the rules used by human schedulers in job shop scheduling were difficult to encapsulate in simulation models. VIS offered a viable alternative by allowing complex decisions to be made externally. Second, VIS are useful in studying the effectiveness of real-time, human decision making in complex systems. Dunkler et al. (1988), for example, used an interactive simulation of a flexible manufacturing system and compared the effectiveness of various automatic scheduling strategies with that of human scheduling in expediting parts through the system. Third, the display of the simulated system in VIS can be visually appealing and can increase effective communication between a manager and the simulation an-

alyst in model development (Bell, 1991; Bishop & Balci, 1990). Fourth, the dynamic visual representation in VIS can highlight logical inconsistencies in the model and can therefore be effective in model verification and validation. Finally, since the user of VIS actively participates in the execution of the simulation, there is potential for increased user confidence in applying the results of the simulation (Kirkpatrick & Bell, 1989).

O'Keefe (1987) outlines different perspectives of visual interactive simulations: statistical, decision support, and simulator. Under the statistical view, there is little or no user interaction with the simulation model during program execution. The interfaces under this mode are primarily for post-simulation animation or performance analysis (Bishop & Balci, 1990). Under the decision support perspective, emphasis is placed on what-if analysis by the user. A user can evaluate alternate scenarios through interaction with the simulation model. The interaction can be user initiated or model prompted. Prompting the user to make a scheduling decision is an example of model-prompted interaction. A situation where a user observes a critical situation in the simulated system and dispatches a constrained resource during the execution of the simulation program is an example of user-initiated interaction. Under the simulator view, interactive simulation architectures can be used to develop human-in-the-loop simulations. Such simulators can provide a powerful synthetic environment for training of human operators in complex systems. Interaction with a high-fidelity synthetic representation of the system can be effective in enhancing user understanding of the complexities of the large, dynamic system.

The major challenges in developing interactive simulations are problems associated with computer hardware and software (Bell & O'Keefe, 1987). Bell (1991) highlights the historic struggle of the early VIS development efforts with advances in computer hardware. Early VIS systems including `See-Why` were developed for large main frames. Currently, personal computers and workstations have become the standard for systems development. Most VIS packages currently available are still hardware dependent and suffer from problems of portability.

Interactive simulations suffer from software problems as well. Several early interactive simulation packages were developed in FORTRAN (e.g., `FORSSIGHT`). Developmental interest has moved towards C and recently towards object-oriented languages (e.g., `ProfiSEE` in Smalltalk-80 [Vaessen, 1989]). While object-oriented programming offers many advantages for simulation modeling in terms of modularity, software reuse, and natural mapping with real world entities (Narayanan et

al., 1996), their application to developing interactive simulations has not been widespread (Bell, 1991). The software to display the simulation model and to facilitate user interaction are often embedded in the simulation model. Such tight integrations make it difficult to maintain large simulation programs and pose limitations in the development of multiple interfaces to a simulation model. The coupling of the simulation model with the interface also makes it difficult for the concurrent development of simulation models and their user interface.

This interactive analyzer component of our system contains a Java-Based Architecture for Developing Interactive Simulations (JADIS). JADIS is hardware independent. JADIS uses the Model-View-Controller (MVC) paradigm from Smalltalk (Goldberg, 1990). The simulation model allows multiple interfaces which are separate processes that execute concurrently on different machines. JADIS integrates concepts from object-oriented programming, concurrent processing, and graphical user interfaces (GUI) to provide a powerful design approach to interactive simulations.

Genetic Algorithms & Interactive Optimization

Our study presented an interesting application for GAs. Traditionally, GAs have been applied for difficult function optimization and control applications (Goldberg, 1989). GAs are computationally simple yet powerful in their search for improvement of solutions. They are not limited by restrictive assumptions about the search space by features such as continuity, existence of derivatives and related notions. For a mathematically complex problem such as aircraft repair time analysis, a GA-based approach to explore the solution space and prune the space into a few good alternatives is appealing. There is an increasing concern in the operations research community on the limitations of mathematical models in providing *the answer* to complex problems where the answer provided may be a result of unrealistic assumptions (Brill et al., 1990). Several researchers believe that the role of models should be to provide "intuition, insight, and understanding that supplements that of the decision makers."

Interactive optimization exploits the user's ability to address the difficult-to-quantify issues, while utilizing the computer to perform the necessary complex numerical calculations (Nulty & Ratliff, 1991). The fundamental components of an interactive optimization system are: (1) models that aid in the solution process, (2) the methodology of user interaction, and (3) the interactive interface between user and the computer. In the modeling component, Nulty and Ratliff (1991) use an integer programming formulation to the problem of Naval fleet scheduling. In their approach, mathematical

algorithms generate a solution to the problem. Then user then uses a graphical interface to tune the solution by modifying parameters to the solution. The algorithms are then reapplied to the entire problem or pieces of the solution. The process is repeated until a satisfactory solution is obtained. The approach we have developed has several interesting similarities and a few major differences. First, the underlying model in the solution generation is based on genetic algorithms. Consequently, the goal is to generate a set of good feasible solutions to the problem in contrast to the generation of the answer to the problem. While the mathematical modeling approach may be more suitable for certain problem which lend themselves to an analytical formulation, our approach is particularly useful in situations where a mathematical formulation is very difficult. Our approach lends itself to generation of multiple alternatives. The human analyst has the capability of examining several different alternatives in detail during the interactive analysis phase. Brill et al., (1990) provide an excellent review of human decision making literature relevant to the process of generation of hypothesis and alternative selection for ill-defined problems. Empirical studies in human-machine decision making have shown that humans perform well when presented with a few, different alternatives than when presented a homogeneous set of alternatives (or a single alternative) as might result during sensitivity analysis in math programming methods (Brill et al., 1990).

CONCLUSIONS

We outlined an approach integrating simulation, evolutionary learning, and human interaction to support generation of satisfactory solutions to complex problems in airbase logistics planning. Our approach is embedded in an architecture which has two major components: (1) a solution explorer, and (2) an interactive analyzer. The solution explorer *generates good feasible design alternatives*, while the interactive analyzer assists in the *selection of the "best" feasible alternative* by providing a synthetic environment for what-if analysis. The architecture is implemented in Java and is portable across computing platforms. Our architecture is useful for simulation, solution exploration, search space pruning, and evaluation of solutions in multi-objective problems.

Preliminary results of the application of the approach to an aircraft repair time analysis problem are promising. The solution explorer eliminated several inferior design solutions and was able to generate more solutions of better quality during the GA-based iterative process. Additional application of the architecture to real airbase logistic problems should provide further insights on the effectiveness of our integrative approach. An obvious future step is to enhance the scale of the architecture

for a real aircraft maintenance problem and document the strengths and drawbacks of our approach. The effectiveness of the interactive analyzer should also be formally evaluated.

In summary, we have demonstrated how simulation, evolutionary learning, and human interaction can be integrated to form a useful tool in solving ill-defined problems in aircraft repair time analysis. Our approach is particularly useful for situations involving multiple objectives where the problem objectives and constraints can not be easily represented in a purely analytical approach. The role of the simulation is to instantiate a descriptive model of the system being studied and to evaluate designs, the GA-based iterator performs random, but directed search based on genetic operators such as reproduction, crossover, and mutation to generate good solutions to the problem, the human is involved in parameter specification and detailed interactive analysis to select among a set of alternative solutions. Future work will involve a formal evaluation of the architecture for realistic data, extension of the architecture to related problems, refinement of the genetic algorithm iterator, and enhancement of visualization capabilities in the interactive analyzer.

BIBLIOGRAPHY

[1] Adiga, S. & Glassey, C. R. (1991). Object-oriented simulation to support research in manufacturing systems. *International Journal of Production Research*, 29 (12): 2529-2542.

[2] Ammons, J. C., Govindaraj, T., & Mitchell, C. M. (1988). A supervisory control paradigm for real-time control of flexible manufacturing systems. *Annals of Operations Research*, 15: 313-335.

[3] Arango, G. & Prieto-Diaz, R. (1991). Domain analysis: Concepts and research directions. In *Domain Analysis and Software Systems Modeling*, (eds.) R. Prieto-Diaz & G. Arango. IEEE Computer Press, p. 12.

[4] Bell, P. C. (1991). Visual interactive modelling: The past, the present, and the prospects. *European Journal of Operational Research*, 54, 274-286.

[5] Bell, P. C. & O'Keefe, R. M. (1987). Visual interactive simulation - History, recent developments, and major issues. *Simulation*, 49(3):109-116.

[6] Bishop, J. L. & Balci, O. (1990). General purpose visual simulation system. *Proceedings of the 1990 Winter Simulation Conference*. 504-512.

[7] Boyle, E. (1990). LCOM explained. Technical report, *AFHRL-TP-90-58*, Air Force Human Resources Laboratory, Wright-Patterson Air Force Base, OH.

[8] Brantley, P., Fox., B. L., & Schrage, L. E. (1987). *A Guide to Simulation*, 2nd Edition, Springer-Verlag, New York.

[9] Brill, Jr., E. D., Flach, J. M., Hopkins, L. D., & Ranjithan, S. (1990). MGA: A decision support system for complex, incompletely defined problems. *IEEE Transactions on System, Man, & Cybernetics*. 20 (4): 745-757.

[10] Carrico, T., Clark, P. K., Shute, N. J., and Zahn, E. A. (1995). Integrated Model Development Environment (IMDE) multi-function aerospace support system. Technical report, *AL/HR-TR-1995-0186*, Armstrong Laboratory, WPAFB, Dayton, OH.

[11] Carrico, T. & Clark, P. K. (1995). IMDE support for Air Force logistics. Technical report, *AL/HR-TR-1995-0187*, Armstrong Laboratory, WPAFB Dayton, OH.

[12] Dunkler, O., Mitchell, C. M., Govindaraj, T., & Ammons, J. C. (1988). The effectiveness of supervisory control strategies in scheduling flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-18, 223-237.

[13] Fishwick, P. A. (1996). Web-based simulation: some personal observations. To appear in *the Proceedings of the 1996 Winter Simulation Conference*, San Diego, CA.

[14] Gobbetti, E. & Turner, R. (1991). Object-oriented design of dynamic graphics applications. *In New Trends in Animation and Visualization*, N. M. Thalmann and D. Thalmann Eds., Wiley Professional Computing, Sussex, England, 43-58.

[15] Goldberg, A. (1990). Information models, views, and controllers. *Dr. Dobb's Journal*, July, 54-61.

[16] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA.

[17] Hurrion, R.D. (1980). An interactive visual simulation system for industrial management. *European Journal of Operational Research,* 5(2): 86-94.

[18] IEEE Standard for Information Technology. (1993). Protocols for distributed simulation applications: Entity information and interaction. *IEEE Standard 1278-1993*. New York: IEEE Computer Society.

[19] Kirkpatrick, P.F. & Bell, P.C. (1989). Visual interactive modeling in industry: Results from a survey of visual interactive model builders. *Interfaces,* 19, 71-79.

[20] Korson, T. & McGregor, J. D. (1990). Understanding object-oriented: a unifying paradigm. *Communications of the ACM*, 33 (9), 40-60.

[21] Krasner, G. E. & Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in Smalltalk 80. *Journal of Object Oriented Programming,* August/September, 26-49.

[22] Law, A. M. & Kelton, W.D. (1991). *Simulation Modeling and Analysis*, 2nd Edition, McGraw-Hill, New York.

[23] Lemay, L. & Perkins, C. L. (1996). *Teach yourself Java in 21 days*. Sams.Net Publishing.

[24] Lyu, J. & Gunasekaran, A. (1993). Developing a visual interactive simulation model for flexible manufacturing systems. *International Journal of Operations and Production Management,* 13(6): 59-67.

[25] Lu, S. C.Y., Tcheng, D. K., & Yerramareddy, S. (1991). Integration of simulation, learning, and optimization to support engineering design. *Annals of the CIRP*, 40(1), 143-146.

[26] Martin, J. & Odell, J. J. (1995). *Object-oriented methods: A Foundation.* Prentice Hall, New Jersey.

[27] Mollamustafaoglu, L., Gurkan, G., & Ozge, A. Y. (1993). Object-oriented design of output analysis tools for simulation languages. *Simulation*. January, 6-15.

[28] Mize, J. H., Bhuskute, H. C., Pratt, D. B., & Kamath, M. (1992). Modeling of integrated manufacturing

systems using an object-oriented approach. *IIE Transactions*, 24 (3), 14-26.

[29] Narayanan, S., Bodner, D. A., Mitchell, C. M., McGinnis, L. F., Govindaraj, T., & Platzman, L. K. (1992). Object-oriented simulation to support modeling and control of automated manufacturing systems. In *Proceedings of the 1992 Western Multiconference*, San Diego: Society for Computer Simulation, 55-63.

[30] Narayanan, S., Bodner, D. A., Sreekanth, U., Govindaraj, T., McGinnis, L. F., & Mitchell, C. M. (1996). Research in object-oriented manufacturing simulations: An assessment of the state of the art. To appear in *IIE Transactions*.

[31] Narayanan, S., Schneider, N. L., Patel, C., Reddy, N, Carrico, T. M., & DiPasquale, J. (1997). An object-based architecture for developing interactive simulations using Java. Submitted to The *Simulation* journal.

[32] Niemeyer, P. & Peck, J. (1996). *Exploring Java.* O'Reilly & Associates, Inc.

[33] Nulty, W. G., & Ratliff, H. D. (1991). Interactive optimization methodology for fleet scheduling. *Naval Research Logistics*, 38, 669-677.

[34] O'Keefe, R. M. (1987). What is visual interactive simulation? (and is there a methodology for doing it right?). In *Proceedings of the 1987 Winter Simulation Conference*. A. Thesen, H. Grant, & W. D. Kelton (Eds.), IEEE, Piscataway, NJ, 461-464.

[35] Paul, R. J. (1989). Visual simulation: Seeing is believing. In *Impacts of Recent Computer Advances on Operations Research.* R. Sharda, B. L. Golden, E. Wasil, O. Balci, and W. Stewart, Eds. Elsevier Science Publishing, New York, NY, 422-432.

[36] Mathewson, S. C. (1984). The application of program generator software and its extensions to discrete event simulation modeling. *IIE Transactions.* 16(1): 3-17.

[37] McGregor, D. R. & Randhawa S. U. (1994). ENTS: An interactive object-oriented system for discrete simulation modeling. *Journal of Object-Oriented Programming,* January, 21-29.

[38] Popken, D. A. (1992). An object-oriented simulation environment for airbase logistics. *Simulation.* 59(5): 328-338.

[39] Rooks, M. (1993). A user-centered paradigm for interactive simulation. *Simulation,* 60(3): 168-177.

[40] Shan, Y-P. (1990). MoDE: A UIMS for Smalltalk. University of North Carolina, Chapel Hill, *TR90-017,* DTIC.

[41] Zeigler, B. P. (1991). Object-oriented simulation with hierarchical, modular models: Intelligent agents and endomorphic systems, San Diego: Academic Press.

APPENDIX

Glossary of Commonly Used OOP Terms (Korson & McGregor, 1990).

Data encapsulation: Describes the hiding of data structures and the implementation of procedures called methods to operate on the data of an object. In Java, the `class` construct is used for data encapsulation. The class combines the structure and behavior of similar objects in its representation.

Inheritance: A technique for deriving new classes from existing ones through subclassing. A subclass inherits both data and methods of the super class (called parent).

Reuse: The ability to use the same software elements for several purposes in different applications.

Polymorphism: The ability to take more than one form. Through polymorphism, the same method results in different behavior depending on run-time binding of objects.

Object: An entity which is a realization of a class, also called as an instance of a class.

Method: A function or procedure associated with a class.